高等学校试用教材

程序设计方法学教程

合肥工业大学　刘宗田　编著

机械工业出版社

高等学校试用教材

程序设计方法学教程

合肥工业大学 刘宗田 编著

机械工业出版社

前　言

程序设计方法学是最近20余年内形成并迅速发展起来的新兴学科。它的研究成果表明，程序设计不是单纯的技巧性活动，而是有自己的规律可循。形式数学推理对开发正确的程序起着积极的指导作用，一些形式化的程序开发技术已经影响了程序设计语言和程序开发工具的发展。学习这门学科，不仅对于研究程序开发技术的专业人员十分必要，而且对于提高程序员的水平和素质具有重要的理论指导意义。

编者在从事这方面教学和科研的基础上编写了这本教材。第一稿于1985年2月完成，第二稿于1986年4月完成，并于同年在长沙召开的全国计算机研究生教学研讨会上展出，受到与会者好评。1988年12月，在全国高等工业学校机电类、兵工类专业教学指导委员会计算机及应用专业教学指导小组第二次会议上，这本教材被选为本科生及研究生教材，决定正式出版。

出版前，编者根据几年来对本科生和研究生的教学经验，对教材做了较大的修改和补充，并调整了章节安排。

本教材主要包括结构程序设计、程序正确性证明、不动点理论、规范技术与抽象数据类型、程序推导、逐步求精和程序变换等内容。每章后面附有一定数量的习题，这些习题大多数都是精心挑选出来的，并由教学实践证明是适用的。

使用这本教材教学，大约需要40～60学时。

姜川同志参加了教材编著及绘图等工作。

张莫成教授和孙怀民教授在教材编写中给予了很大的帮助。冯玉琳教授耐心审阅了全稿并提出许多宝贵建议。宋宝珍副教授等对教材的编写、修改及出版给予了热情的关心和支持。在此，特向他们表示衷心的感谢。

由于水平限制，教材中难免存在错误和不妥之处，恳请同行专家及广大读者批评指正。

编　者

992年1月

目　录

前言

绪论 …… 1

一、程序设计的短暂历史回顾 …… 1

二、结构程序设计 …… 1

三、证明程序正确 …… 2

四、构造正确程序 …… 3

第一章　结构程序 …… 4

第一节　流图程序 …… 4

一、程序的有向图表示 …… 4

二、真程序 …… 5

第二节　程序函数 …… 5

一、执行图和执行树 …… 5

二、程序函数 …… 6

第三节　结构定理 …… 10

一、素程序 …… 10

二、复合程序 …… 10

三、结构定理 …… 11

四、递归结构程序 …… 12

第四节　层次化控制结构 …… 14

一、while型程序结构 …… 14

二、PDL结构 …… 14

三、程序非正常退出和循环非正常退出 …… 15

四、程序的结构化变换 …… 16

第五节　读结构程序 …… 18

一、程序阅读方法 …… 18

二、读素程序 …… 19

三、逐步抽象阅读 …… 21

四、结构程序的逻辑注释 …… 24

习题 …… 25

第二章　程序验证——归纳断言法 …… 30

第一节　基本概念 …… 30

一、程序验证 …… 30

二、规范问题 …… 30

三、程序正确性 …… 30

四、中间断言 …… 31

第二节　反向代换技术 …… 31

一、路径函数 …… 31

二、反向代换 …… 31

三、路径部分正确性 …… 32

第三节　部分正确性证明的归纳断言法 …… 33

一、部分正确性的归纳断言定理 …… 33

二、部分正确性证明算法 …… 34

三、数组对程序验证的影响 …… 37

第四节　终结性证明 …… 41

一、良基集合 …… 41

二、良断言与良函数 …… 41

三、良基集合法 …… 42

习题 …… 44

第三章　程序验证——公理化方法 …… 46

第一节　Hoare的公理化方法 …… 46

一、归纳表达式 …… 46

二、验证规则 …… 46

三、验证规则定理 …… 47

第二节　终结规则法 …… 49

一、终结规则 …… 49

二、终结规则定理 …… 50

第三节　谓词转换器 …… 53

一、一种非确定性程序设计语言 …… 53

二、最弱前置条件 …… 54

三、程序证明的演绎系统 …… 55

四、终结性和正确性验证 …… 56

习题 …… 58

第四章　程序验证——不动点方法 …… 60

第一节　函数 …… 60

一、定义域和值域的扩充 …… 60

二、单调函数 …… 60

三、自然扩充 …… 61

四、单调函数的复合 …… 62

五、最小上界 …… 62

第二节　泛函 …… 63

一、单调性与连续性 …… 63

二、泛函的不动点 …… 64

第三节　递归程序……65
一、计算法则……65
二、不动点计算法则……67
第四节　递归程序的验证方法……67
一、逐步计算归纳……67
二、完全计算归纳……69
三、不动点归纳……71
四、结构归纳……73
习题……75
第五章　构造正确程序……79
第一节　利用不变式构造程序……79
第二节　不变式的推导技术……84
第三节　程序设计演算……87
习题……91
第六章　形式规范技术……93
第一节　规范语言……93
一、规范与程序……93
二、规范语言的要求……93
三、规范语言的一个实例……94
四、存在性和唯一性……96
五、哲学家问题的规范……96
第二节　抽象数据类型……98
一、类型抽象……98
二、抽象类型规范方法分类……98
三、Hoare方法……99
四、代数方法……101
五、代数规范的一致性和完备性……104
习题……105
第七章　逐步求精开发程序……106
第一节　逐步求精方法……106
第二节　非递归程序的开发……106
第三节　递归程序的开发……116
第四节　逐步求精方法总结……120
习题……121
第八章　程序变换……122
第一节　基本概念……122
一、程序等价……122
二、程序段、程序图式和图式变量……122
三、程序变换规则概述……123
四、程序变换的基本原理……123
五、程序变换语言……123
第二节　基本变换规则……124
一、扩展(unfold)和折叠(fold)……124
二、定义(definition)规则……124
三、取样(instantiation)规则……125
四、定律(law)……125
五、抽象(abstraction)规则……125
六、改写规则……125
第三节　由谓词定义式向函数定义式的变换……125
第四节　函数定义级上的变换……128
一、减少递归函数的应用性表现的变换……128
二、把一些递归变换成尾递归……129
第五节　函数定义级向过程级变换……130
第六节　过程级变换……131
第七节　程序变换系统……131
习题……132
参考文献……133

绪　论

程序设计，实际上包括问题需求分析、规范描述、过程设计、实现、连接、验证、调试和评价等阶段。有关需求分析、连接、调试和评价方面的内容，可以在软件工程方面的书籍中找到。因此，我们这本教材主要讨论程序的规范描述、过程设计、实现和验证，而且主要以小程序为例进行叙述。

为什么以小程序为例讨论程序设计呢？经验表明，生产小的正确的程序本身就是一项困难的任务。当小程序开发还弄不懂时，怎能指望有效地开发大程序呢！

一个大的程序或软件，最终是由 n 个小程序或程序"模块"构成。假定这些互相独立的模块的每一个正确概率是 p，则整个系统的正确概率 P 只能满足 $P \leqslant p^n$。当 n 很大时，为了保证整个系统可靠，则 p 必须接近1。

一、程序设计的短暂历史回顾

电子计算机问世以后的最初几年，计算机价格昂贵，功能有限，而且不太可靠。当时使用计算机，主要是保持它有秩序的工作，并尽可能有效地使用它。程序员的任务是用特定计算机的机器语言编写简单的（从今天的角度看）小的算法，使用尽可能巧妙的手段和技术，以克服内存和速度上的限制。当时的程序，完全是私人的东西，其它人很难读懂。程序是为特定机器、特定目的而编写的，很难为了别处的使用而转输给另外的机器。

FORTRAN及其它高级语言的出现，使程序设计方法有了一些改变，但主要思想仍然是压缩使用计算机的内存和时间。

后来，由于计算机变得更有效和更灵活，硬件成本已急剧下降，而且程序员要解决的问题变得更加复杂，这时，程序员发现，可以使用的巧妙方法远远不够了，越来越多的时间花费在程序的调试中。软件花费超出预算已成为寻常之事，许多软件尚未出产就不得不宣布报废。因此，计算机发展的重点开始从硬件转向了软件。

程序设计的形势已经恶化，致使北大西洋公约组织在1968年和1969年召集会议，讨论如何以合理的花费生产可靠软件的问题。

虽然不是所有与会者都同意使用"软件危急"名词，但大家一致认为，不知道如何用合理的方法生产软件。今天，我们知道，程序设计是一项困难的任务，程序设计方法学方面的大部分研究工作还正在进行，这些研究工作在程序设计领域已取得了可喜的成绩。

二、结构程序设计

"结构程序设计"术语是Dijkstra在他的专题论文"Notes on Structured Programming"〔1972年〕中首先使用的，它震动了整个程序设计界。就其狭义观念（不使用goto）而言，它已得到大多数人的响应；就其广义观念而言，它已经影响了程序设计方法学的研究和实践。

人们可能要问，是否程序员能很自然地采用结构程序设计方法？回答是否定的，这可由下述三个事实看出。第一，中等水平的程序员在合理的时间内完不成自己的任务；第二，最终的程序不好懂，别人很难读和修改它；第三，大多数程序都不满足原来的规范，并且充满了错误，其中有一些甚至几年都未被发现。

一些重要的研究是为了解决下述问题：

①程序研制过程应当如何组织？

②程序应当如何组织？

③怎样知道一个程序是正确的？

④文件应当如何写，以便更好地描述程序？

⑤应当研制什么样的软件工具以支持开发可靠程序？

（一）程序的规模与复杂性问题

程序设计是一项困难的智力任务，这是由于所处理的问题的规模和复杂性造成的。规模确实是一个因素，编译程序有5000到50000行高级语言代码，而操作系统甚至可以达到数十万行，在这种情况下，任何个人只能记住甚至只能读出该系统的一小部分。

但是，程序规模不是唯一的原因。一个五六行的程序，在不好好组织和注释的情况下也可能很难理解。

程序员面临规模与复杂性问题，因此存在着内容的量与他能接受的复杂性的限度之间的矛盾。程序员必须首先认识他的限度，而不是忽视它。

明智的程序员应当很好地组织自己的任务，使混乱的内容成为易懂的程序，层次化结构的思想是处理任何复杂问题的有效方法。程序员应当找出好的表示形式，设法简化程序的复杂性，使当前所处理的部分在他可接受的限度之内。

（二）支持程序设计的数学推理方法

Dijkstra〔1972年〕讨论了三种推理方法：枚举推理、数学归纳和抽象。

使用枚举推理，可以理解顺序语句、条件语句的使用。试图通过判定每条可能的执行路径都能正确地工作而断定整个程序正确，这只能在路径数有限且较少的情况下方能适用。但将程序划分为有限个小的单元的方法是存在的，例如用割断循环的方法得到有限条不含中间割点的路径，又如限定程序仅含有顺序、分支和循环结构，将程序划分为数个结构上独立的单元。

数学归纳用以理解循环和递归结构。典型的循环可以重复零次、一次、二次或任意次。用归纳考查循环正确，如同用归纳法证明整数的性质一样。

抽象可看作为将来使用的目标抽出一个或多个特征或性质，它的目的是只涉及事物的有关性质而忽略无关性质。抽象伴随整个程序设计过程。变量就是它的当前值的抽象形式。当我们写一个过程，然后写几个调用时，就使用了抽象。这就是，当写过程时，涉及它如何做，然后完全忘记这个如何，而仅涉及它做什么。抽象数据类型是对数据及其上面一些特定运算的抽象。

如果枚举推理、归纳和抽象是主要的推理方法，那么就应当限制自己，以构造和组织能有效地使用这些方法的软件产品。结构程序设计就是这类限制方法的一种。有关结构程序的基本概念和基本原理，将在第一章中介绍。

三、证明程序正确

历史上对待程序出错的观念是不正确的。过去认为，错误是必然的，找出它们并确定它们的位置，很自然地要求程序员花费一定的时间（约占整个程序设计时间的30%～60%）。这样就引出“调试”术语。疵点象蚊子一样，总是存在，并且只能在发现时拍打。

Dijkstra指出，程序调试不能表明错误不存在（不存在错误是我们希望的）。而只能表明错误存在。另外，已注意到，在调试程序中，后期发现错误，要确定其位置，花费的时间是相当多的，因此错误应当尽早发现，或者根本不引进程序中。

这样，虽然还要调试，但程序员在调试期间发现错误是例外情况，而不是必然规律。他应当在调试之前就知道他所研制和组织的程序是正确的。

定理证明是使读者确信该定理是正确的这样一个论证过程。证明可以是形式的，也就是根据公理一步步地应用推理规则。另一方面，它也可以是完全非形式的说理。

经验表明，常用的非形式的说理，用于程序，是不充分的。特别是对于大的复杂问题。因此，我们必须采用更系统的形式技术。

证明程序的思想酝酿了很长时间，例如McCartry于1961年曾提出："人们不应当试图调试程序直至无错，而应当证明它有所考虑的性质"。Naur于1966年强调了程序证明的重要性，并提出了一种非形式化的技术。Floyd建议，证明技术应当为程序设计语言提供一个合适的语义定义，并提出用程序加中间断言的方法证明程序。在Floyd的思想影响下，Hoare为计算机程序提出了第一个公理方法。Floyd的归纳断言法以及Hoare和Dijkstra的公理化方法将在第二章和第三章介绍。

递归程序的性质，可以用不动点理论分析。程序的最小不动点，可看作它所定义的计算解函数。证明递归程序正确，实质上就是证明它的最小不动点与预期的函数一致。有关不动点理论，将在第四章介绍。

四、构造正确程序

把证明程序正确性的思想引入程序构造过程，就产生了边构造边证明其正确性的方法。至于证明的思想如何引导程序推导，目前还不十分清楚，但由程序规范引导不变式，由不变式引导编写循环程序，是有规律可循的。Dijkstra提出了一种更形式的方法，使用谓词转换器及其演算规则集合，可以推导出正确的程序。这方面的内容将在第五章介绍。

由Wirth首先提出的逐步求精研制程序的方法，是结构程序设计思想的继续。形式地推导正确程序的方法，对于较小规模的程序是有效的，但用以处理较大规模的程序设计，则有困难。而逐步求精方法虽不及演算推导方法严谨，但它可以用以处理较大规模的程序。实际上，逐步求精的思想适合处理任何复杂系统。即便是小规模的程序，在演算推导过程中仍然需要采用逐步求精方法。第七章专门介绍逐步求精开发程序的方法，并且选用了具有一定规模的程序作为程序开发实例。

构造正确性程序的另一种方法，是程序变换，它对程序规范应用一连串的保护正确性的变换规则，最终得到可执行的程序。程序变换的主要目的，是用计算机承担程序设计中最繁琐的一些工作。它是程序设计自动化的很有希望的途径之一。有关程序变换的内容可阅读第八章。

对于许多问题，构造递归程序要比构造非递归程序容易。但从运行角度衡量，递归程序远不及同一问题的非递归程序效率高。因此，把递归形式看作程序规范与程序之间的中介，是有一定道理的。由递归转换为循环结构很有意义。在程序变换中，递归形式起着重要的作用。不动点理论不仅可以用以证明递归程序性质，而且可以引导构造递归程序。在逐步求精一章中，我们用了相当篇幅介绍逐步求精构造递归程序的方法。

构造正确性程序还涉及许多其它方面的问题，其中主要的是程序规范、抽象数据等。第六章专门介绍这方面的内容。

人工智能领域中的近期成果已逐渐用于自动程序设计研究。程序综合本身就被列入人工智能范畴。在程序变换中应用知识工程原理和产生式系统的搜索技术，已取得了可喜成绩。

第一章 结构程序

结构程序设计的思想是由E·W·Dijkstra在70年代初提出来的。他的这方面的论文和学术报告一发表，立刻引起计算机科学界的重视，同时也引起一场争论。争论的焦点不是结构程序设计的思想是否正确，而是goto语句是否有害和是否必须废除的问题。D·E·Knuth在以Dijkstra为首的反对goto语句一派和坚持维护goto语句的另一派之间取折衷态度，发表了“带goto的结构程序设计”一文，才使这场争论平息下来。无论如何，结构程序设计目前已成为计算机界普遍接受的思想。

本章主要介绍结构程序的性质、结构定理及结构程序的阅读理解方法，并以流图程序作为主要研究对象。

第一节 流图程序

一、程序的有向图表示

流图是有向图，它描绘了程序的执行控制流和被执行的指令。程序的每一指令对应于流图的一个结点，每一可能的控制流向，对应于流图的一条射线。如果某一指令结点有多于一条引出线，则它就是控制指令。如果某一控制指令的执行不影响除指令计数器之外的数据，则它就是纯控制指令，否则该控制指令就有副作用(side effects)。

如果一个流图结点有一条引入线和一条引出线，则称之为**函数结点**，见图1-1。这里有一命名为 f 的函数与该结点联系，它代表一条赋值指令。函数结点这一术语基本上是合适的，因为任一赋值指令对数据的作用完全能用函数描述出来。

图1-1 函数结点

如果一个流图结点有单一的引入线和两条引出线，并且是纯控制指令，则称之为**谓词结点**，见图1-2。这里有一个名为 P 的谓词与该点相联系。谓词结点根据该谓词执行为真或为假(T或F)指出执行的流向，它不对其它数据起作用。

为了使流图更加清晰直观，习惯上引进一种“无操作”的结点。具有两条引入线和一条引出线的“无操作”结点称为**汇集结点**，如图1-3所示。

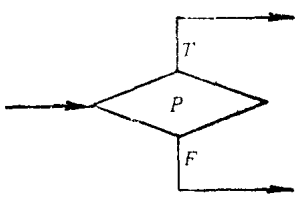


图1-2 谓词结点

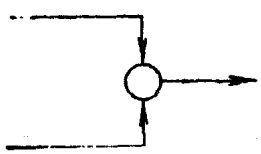


图1-3 汇集结点

流图的控制结构仅保留函数结点、谓词结点和汇集结点的次序特征，而忽略与这些结点相联系的函数、谓词和谓词值的特性。图1-4所示的流图具有名为 p 和 q 的谓词以及名为 g

和 h 的函数，它有图1-5所表示的控制结构。

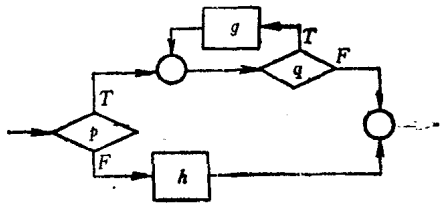


图1-4　具有 p、q谓词和 g、h 函数的流图

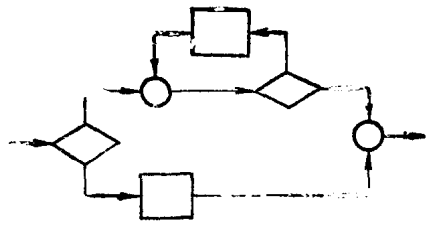


图1-5　流图的控制结构

二、真程序

真程序是具有下述控制结构的程序：

①有单独的入口线和单独的出口线

②对于每一结点，都有从入口线到出口线的一条路径通过。

条件②取缔了象图 1-6 所示的控制结构，这些控制结构有不合法的结点集(见图1-6a)和不可达的结点集(见图1-6b)。

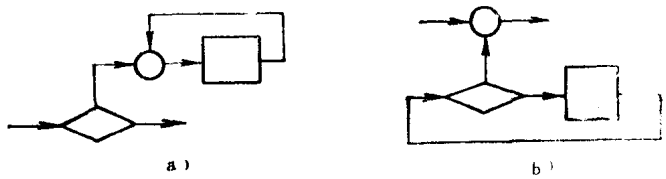


图1-6　不合法的控制结构

一个真程序能抽象为单个函数结点，该函数结点概括了这个真程序对数据的作用。相反，程序的函数结点，也都能扩展为真程序，这种扩展不影响程序其它部分的功能。

程序中自身为真的部分称为**真子程序**。例如图1-4中，由结点 q、g及 q 之前的汇集结点组成的部分是真子程序。g和 h 也分别是真子程序。但 p 和 q 都不是真子程序。

第二节　程 序 函 数

一、执行图和执行树

流图程序定义了沿路径和环路的执行序列，这些执行序列可以用执行图(execution chart，简称E图)和执行树的概念加深理解。

执行图是一有限树。对于任一真程序，可以用下述方法构造出具有与该真程序相同结点和射线的E图：

①开始，E图仅包括流图的入口线和与入口线相连接的谓词、函数或汇集结点。

②考虑由入口线出发的每条执行路径，如果路径中遇到在该路径中过去未发现的函数、谓词或汇集结点，则把该结点的所有引出线和与这些引出线相连的所有结点加到E图的该执行路径上。

③当所有的执行路径在出口线或在该路径先前出现过的结点上终结时，则E图构造完毕。

显然，这个过程以有限树而终结，因为每条路径最多包含该流图的所有结点。例如，图1-7所示的流图产生的E图如图1-8所示。

流图的执行可以用它的E图的执行路径给出。在E图上，当执行到达该路径重复出现的结束点时，应返回到该点先前出现的地方继续执行。显然，当且仅当流图的E图没有重复结点作为执行路径的结束点时，流图才是无循环的。另外，所有汇集结点，除去重复的以外，都可以删除，例如图1-8中的E图可以简化为图1-9中的E图，这里仅有汇集结点2被保留。

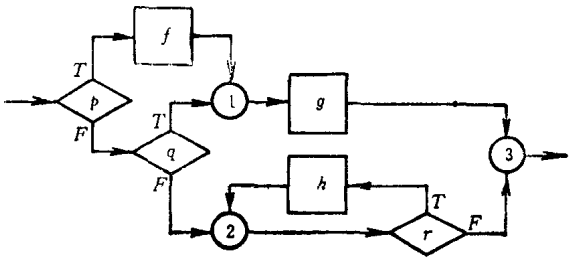


图1-7 流图

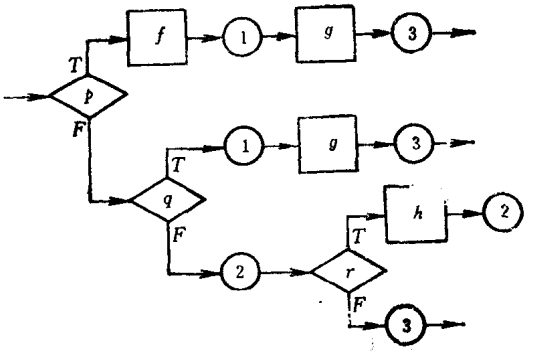


图1-8 E图

流图程序的**执行树**(E树)是这样的树：它的路径描绘流图的所有可能的执行序列而无需返回。如果流图中没有循环，则其相应的E图就是E树。如果流图中有循环，则E树是无限树，它是在E图中，反复用重复结点的第一个结点为根的子树代换后继重复结点而得到的。代换之后，重复结点的第一个出现点可以删去。例如，图1-9E图所产生的E树如图1-10所示。

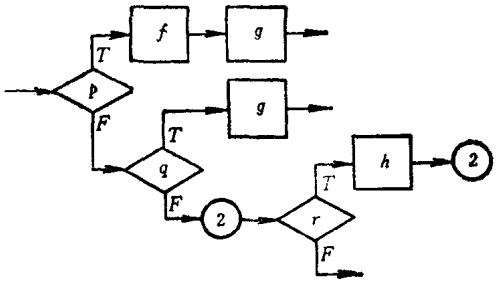


图1-9 简化E图

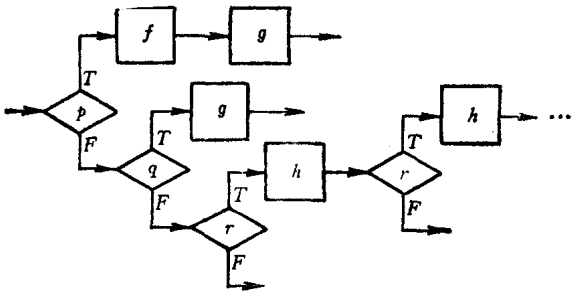


图1-10 E树

二、程序函数

我们可以把程序对数据的作用看作一种映射，是在数据的**状态空间**上，从初始状态到终结状态的映射。如果限定所研究的程序都是确定性的，那么，程序的作用相当于一个函数，因此引出**程序函数**这一概念。

为了介绍程序函数，我们先分析一条简单的赋值语句对数据所起的作用。

假如程序中的数据空间为(x, y, z)，赋值语句

$$x := y$$

是把初始数据状态(x_0, y_0, z_0)转变为终结数据状态(y_0, y_0, z_0)，这一映射可表示为

$$\{[(x, y, z)\ \ (u, v, w)]\mid u=y\wedge v=y\wedge w=z\}$$

或写为

$$\{[(x, y, z), (y, y, z)]\}$$

对于赋值语句，这一映射称为**赋值函数**，用在该赋值语句外加方括号的形式表示。上述赋值函数可表示为

$$〔x:=y〕=\{〔(x,\ y,\ z),\ (y,\ y,\ z)〕\}$$

注意，等式两边的字母有明显不同的含意，左边的x和y是程序变量名，而右边的x，y，z是函数值的名。因此

$$〔x:=y〕=\{〔(u,\ v,\ w),\ (v,\ v,\ w)〕\}$$

与上面等式一致。

赋值语句的右部可以使用函数，注意不要与赋值函数本身混淆。例如

$$x:=\max(y,\ z)$$

使用了如下定义的函数max：

$$\max=\{〔(y,\ z),\ (u)〕|(y\geqslant z\wedge u=y)\vee(z\geqslant y\wedge u=z)\}$$

但赋值函数是

$$〔x:=\max(y,\ z)〕=\{〔(x,\ y,\ z),\ (\max(y,\ z),\ y,\ z)〕\}$$

以变量为下标的数组元素的赋值，应将下标看成数据空间的一部分。例如x是三元素的数组$x(1:3)$，则并行赋值

$$i,\ x(i+1):=x(i),\ i+x(i+1)$$

至少影响四个元素

$$(i,\ x(1),\ x(2),\ x(3))$$

准确地讲，上述赋值可以表示为不同情况下的分别的简单赋值

$$\begin{aligned}&i=1\longrightarrow i,\ x(2):=x(1),\ 1+x(2)\\ |&i=2\longrightarrow i,\ x(3):=x(2),\ 2+x(3)\end{aligned}$$

因此，它的赋值函数只能用条件规则形式给出

$$\begin{aligned}&〔i,\ x(i+1):=x(i),\ i+x(i+1)〕\\ =\{&(t,\ x,\ y,\ z),\ (k,\ u,\ v,\ w)|t=1\longrightarrow k=x\wedge u=x\wedge v=1+y\wedge w=z\\ &|t=2\longrightarrow k=y\wedge u=x\wedge v=y\wedge w=2+z\}\end{aligned}$$

下面我们分析真程序对数据的作用。我们用〔P〕表示程序P的程序函数。

对于单个函数结点$f=\{(x,\ y)\}$的程序P（图1-11），其程序函数是

$$〔P〕=\{(x,\ y)|y=f(x)\}$$

这里x，y应看作数据状态空间矢量。

两个函数结点的顺序结构，如图1-12所示，这里$g=\{(x,\ y)\}$，$h=\{(y,\ z)\}$。它的程序函数是

$$〔P〕=\{(x,\ z)|z=h(g(x))\}$$

本书中常把$h(g(x))$表示为$h\cdot g(x)$的形式。

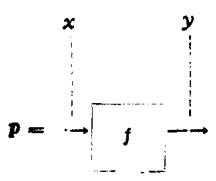


图1-11　单结点程序

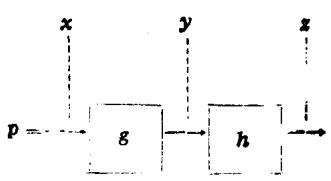


图1-12　两函数结点的顺序结构

无循环程序可以构造出有穷E树。对于E树的每条路径，路径中的函数结点顺序对数据的作用即是该路径对数据的作用。谓词结点与函数结点中的谓词与函数，定义了执行通过该路径的充分必要条件。这样就形成了每一路径的**路径函数**，合并诸路径函数，则得整个程序的程序函数。例如图1-13中的流图程序，它的E图有5个结束点，分别用1～5标出，见图1-14。相应于这5个结束点的5条路径的路径函数是

①$\{(x, y)|p(x)\wedge q(f(x))\wedge y=g(f(x))\}$

②$\{(x, y)|p(x)\wedge\sim q(f(x))\wedge r(h(f(x)))\wedge y=g(h(f(x)))\}$

③$\{(x, y)|p(x)\wedge\sim q(f(x))\wedge\sim r(h(f(x)))\wedge y=h(f(x))\}$

④$\{(x, y)|\sim p(x)\wedge r(h(x))\wedge y=g(h(x))\}$

⑤$\{(x, y)|\sim p(x)\wedge\sim r(h(x))\wedge y=h(x)\}$

程序函数还可以用条件规则定义：

$$\begin{aligned}[P]=\{(x, y)|&(p(x)\wedge q\cdot f(x)\rightarrow y=g\cdot f(x)\\&|p(x)\wedge\sim q\cdot f(x)\wedge r\cdot h\cdot f(x)\rightarrow y=g\cdot f(x)\\&|p(x)\wedge\sim q\cdot f(x)\wedge\sim r\cdot h\cdot f(x)\rightarrow y=h\cdot f(x)\\&|\sim p(x)\wedge r\cdot h(x)\rightarrow y=g\cdot h(x)\\&|\sim p(x)\wedge\sim r\cdot h(x)\rightarrow y=h(x))\}\end{aligned}$$

对于循环程序，我们用f_j定义E图中以第一次出现的重复结点j开始的子E图的程序函数。用单函数结点f_j代换以后出现的重复结点j，得到无循环E图。用构造无循环程序函数的方法构造整个程序的程序函数及每个重复结束点j所对应f_j，这样可得到一组方程。解这方程组，则可表示出$[P]$。

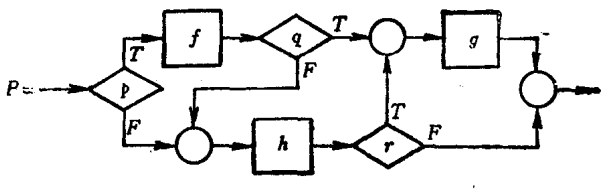


图1-13　流图程序

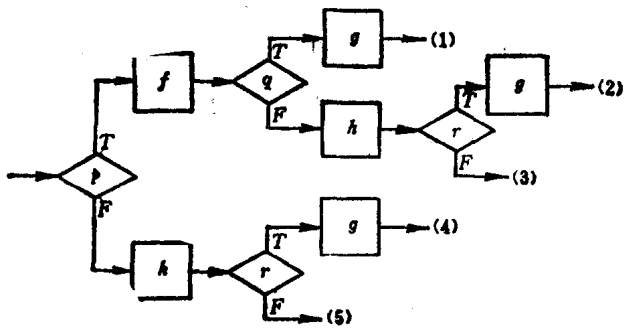


图1-14　图1-13流图程序的E图

例如，图1 15所示的流图程序有图1-16所示的E图。又如图1-17所示，把f_1、f_2、f_3联系到结点1、2、3上，得到f_1、f_2、f_3的函数等式：

$$f_1=\{(x, y)|y=f_2\cdot g_1(x)\}$$

$$\begin{aligned}f_2=\{(x, y)|&(p_1\cdot g_3(x)\rightarrow y=f_1\cdot g_2\cdot g_3(x)\\&|\sim p_1\cdot g_3(x)\rightarrow y=f_3\cdot g_3(x))\}\end{aligned}$$

$$f_3=\{(x,\ y)\mid(p_2(x)\wedge p_3(x)\rightarrow y=f_3\cdot g_5(x)$$
$$\mid p_2(x)\wedge\sim p_3(x)\rightarrow y=x$$
$$\mid\sim p_2(x)\rightarrow y=f_2\cdot g_4(x))\}$$

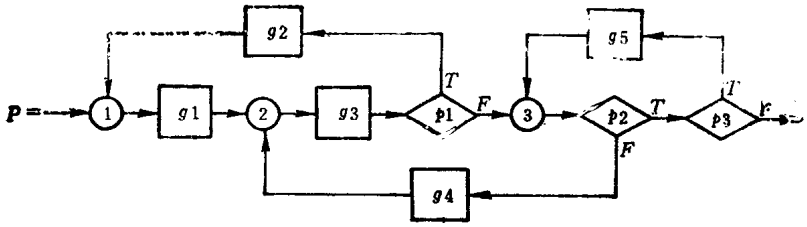


图1-15　流图程序

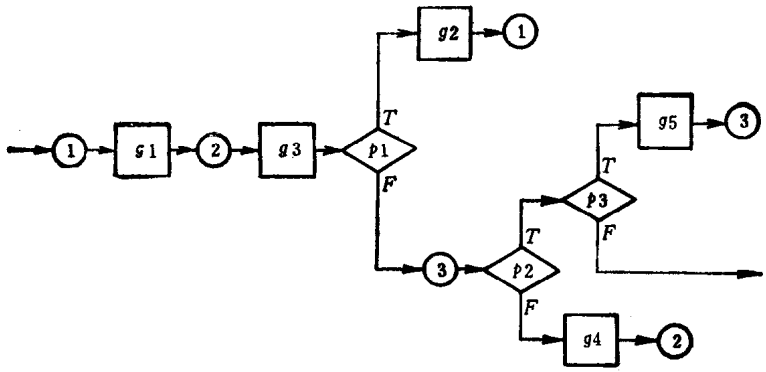


图1-16　图1-15流图程序的E图

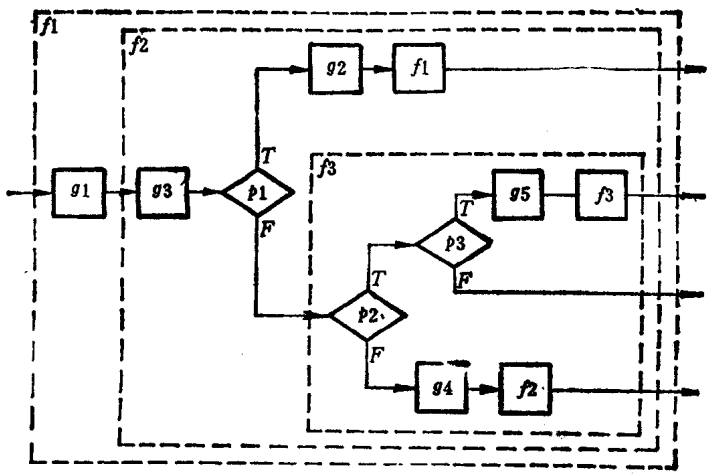


图1-17　代换后的E图

如果这组等式有解，则 P 的程序函数就是

$$〔P〕=f_1$$

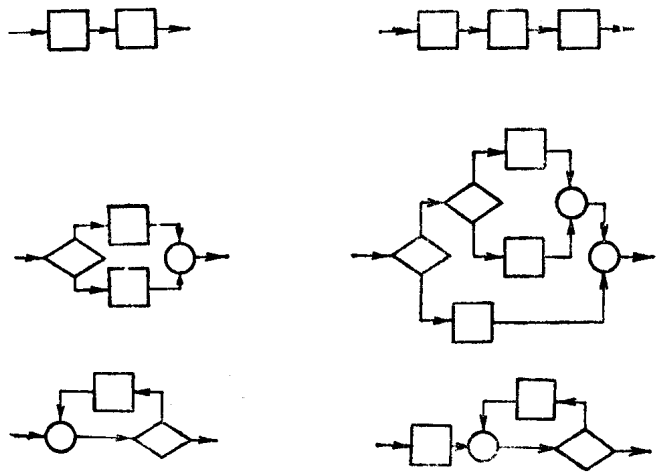
这类方程可以用不动点方法求解。不动点理论将在第四章介绍。

第三节　结 构 定 理

一、素程序

素程序是不存在多于一个函数或谓词结点的真子程序的真程序。例如，具有图1-18a所示控制结构的程序是素程序，而具有图1-18b所示控制结构的程序是真程序，但不是素程序。

真程序可以按结点数（不计算汇集结点数）列举，并能划分为素和非素两类。包含1～4个结点的素程序总共有15个(读者可以逐个画出它们的流图控制结构)，而其中包含一个或多个函数结点的只有7个。不难理解，不含有函数结点的素程序对数据不起作用，因此，这样的素结构在程序中无意义，可以删除。下面我们给出7个含有函数结点的素程序，并为它们取专门的名字。它们可以分别对应过程型程序的某一控制结构，见图1-19。



a）素程序结构　　b）非素程序结构

图1-18　素结构和非素结构
a）素程序结构　b）非素程序结构

为了表示这些素程序的程序函数，我们在函数 g 上加上标表示重复，即

$g^0(x)=x$ 和 $g^k(x)=g\cdot g^{k-1}(x)$ 对于 $k=1, 2, \cdots$。这样，这些程序的程序函数是：

$[f]=f$

$[g;\ h]=\{(x, y)\mid y=h\cdot g(x)\}$

$[\textbf{if}\ p\ \textbf{then}\ g\ \textbf{fi}]=\{(x, y)\mid (p(x)\wedge y=g(x))\vee(\sim p(x)\wedge y=x)\}$

$[\textbf{while}\ p\ \textbf{do}\ g\ \textbf{od}]$

$=\{(x, y)\mid \exists\ k\geqslant 0\ ((\forall j,\ 0\leqslant j<k)(p\cdot g^j(x))\wedge\sim p\cdot g^k(x)\wedge y=g^k(x))\}$

$[\textbf{do}\ g\ \textbf{until}\ p\ \textbf{od}]$

$=\{(x, y)\mid \exists k>0((\forall j, 1\leqslant j<k)(\sim p\cdot g^j(x))\wedge p\cdot g^k(x)\wedge y=g^k(x))\}$

$[\textbf{if}\ p\ \textbf{then}\ g\ \textbf{else}\ h\ \textbf{fi}]$

$=\{(x, y)\mid (p(x)\wedge y=g(x))\vee(\sim p(x)\wedge y=h(x))\}$

$[\textbf{do1}\ g\ \textbf{while}\ p\ \textbf{do2}\ h\ \textbf{od}]$

$=\{(x, y)\mid \exists k\geqslant 0((\forall j,\ 0\leqslant j<k)(p\cdot g\cdot(h\cdot g)^j(x))$

$\wedge\sim p\cdot g(h\cdot g)^k(x)\wedge y=g\cdot(h\cdot g)^k(x))\}$

二、复合程序

我们给出**复合程序**的定义如下：

①素程序是复合程序；

②用复合程序代换素程序中的函数结点，得到的仍是复合程序。

如果限定某些素程序为构造复合程序的基础集合，则可以产生一特定的复合程序类，也就是真程序的一个子集。例如，集合{sequence, if then else}产生无循环程序类，而集合{if then else, while do}产生的程序类，其执行树的任一路径上，最多有一个不同的函数结点(可能重复)。某个复合程序类可能是另一个类的子集，例如，集合{sequence, do until}产生的类是集合{sequence, if then, while do}产生的程序类的子集。

定义：结构程序是由一个固定的素程序基础集合构造的程序。

三、结构定理

在研究结构定理之前，先介绍程序等价的概念。现在我们定义两种程序等价。如果两个程序有相同的执行树，则它们是**执行等价的**；如果它们有相同的程序函数，则它们是**函数等价**的。执行等价蕴含着函数等价，但反之不然。

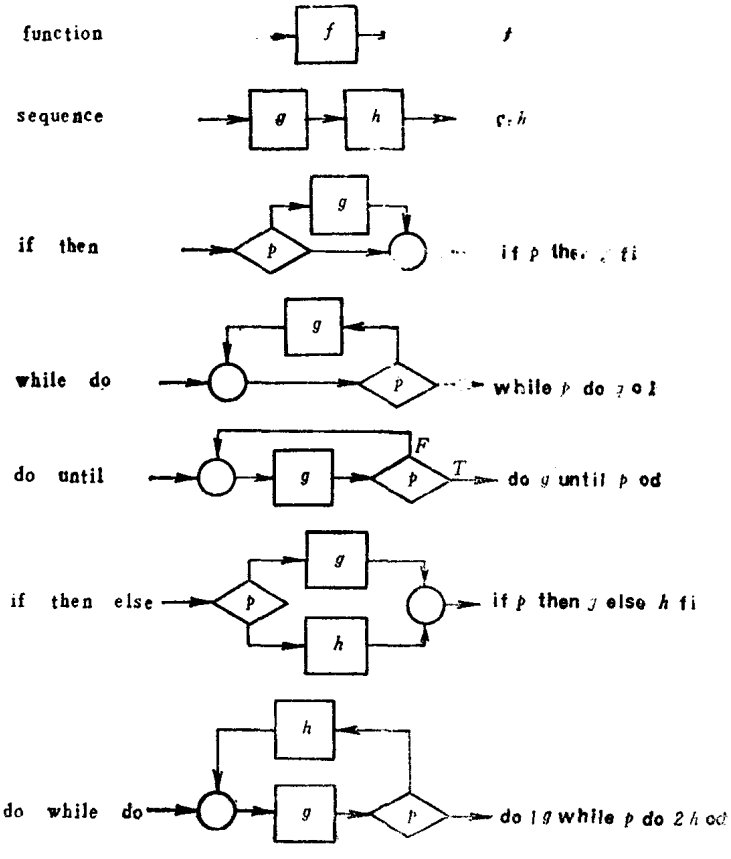


图1-19　7种素程序与过程型程序控制结构的对应关系

结构定理：任一真程序都函数等价于某个具有基础集合{sequence, if then else, while do}的结构程序，而且该结构程序使用原来程序中的函数和谓词，另外还要用到对一个附加计数器的赋值和测试。

证明：考虑任意的真程序，并且对它的函数和谓词结点任意编号1，2，…，n，程序的出口线编号为0。然后对每条射线附上其所达到的最近的那个函数或谓词结点的编号，如果没有这样的结点(即到达出口线)，则附上0。对于编号后的程序作如下的处理：

①对编号为i，引出线为j的每个函数结点，如图1-20，构造一个新的sequence程序g(与引入线i相联)，如图1-21所示。

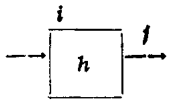


图1-20　函数结点

图1-21　sequence程序g_i

②对每个编号为i的谓词结点(见图1-22)构造新的if then else程序g_i，如图1-23所示。

③构造一个对L赋初值的while do程序Q，L的初值应当是原来程序的入口线的编号。该

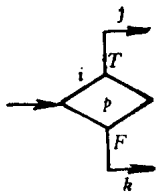


图1-22 谓词结点

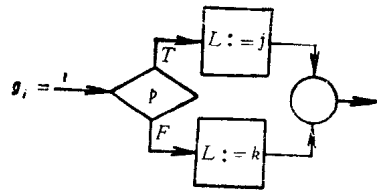


图1-23 if then else 程序g_i

while do部分是由嵌套的对L的从1到n的测试组成，并且每个if then else的真引出线连接g_i。假定程序入口线的编号为1，while do程序如图1-24所示。原来的程序和新构造出的程序是函数等价的，而Q是由基础集合{sequence, if then else, while do}产生的结构程序，因此结构定理得证。

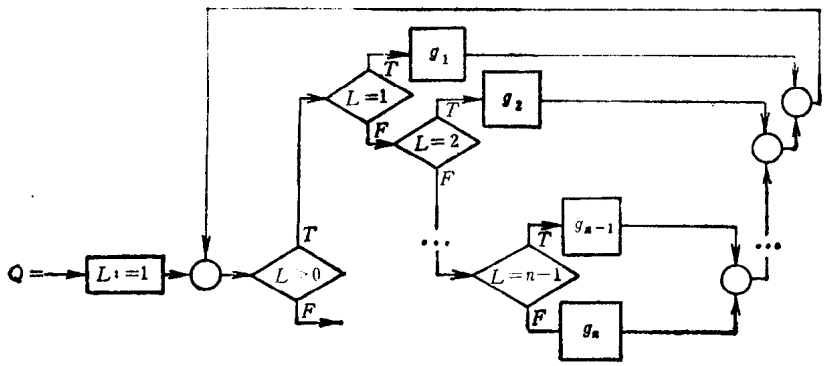


图1-24 while do程序

四、递归结构程序

在前面的结构定理的证明中所产生的程序虽然是结构的，但不太清晰并且效率不高。我们用消除不必要的计数器L及其测试的方法改进这一结构。

其方法是：对于某个给定的$j>0$，用程序g_j代换所有的赋值$L:=j$（注意，对于$L=1$，while do前面的初始化$L:=1$应一起由g_1代换）。因为j值不再赋给L，所以测试$L=j$能从结构中删去（即$L=j$为真时通过的那一分支被删去）。继续这一过程，直到除$L:=0$外，所有对L的赋值都已被删除，或每个剩余的g_i'都包含有赋值$L:=i$，这里g_i'是代换后$L=i$分支上的当前复合程序。

如果程序无循环，则赋值$L:=0$出现在每条通路上，因此计数器L和whiledo循环也可以消除。

用上述方法简化后，剩下来的是一个具有合理清晰性和较高执行效率的复合程序。

例如，图1-25所示的标号结构程序可以改进如下：第一，对$L=3$路径上的$L:=4$用$L=4$路径的程序进行代换，并删除$L=4$路径，得到新的程序如图1-26所示。然后用$L=3$路径的程序代换$L=1$路径中的$L:=3$，并消除$L=3$路径，得程序如图1-27所示。用同样方法代换$L:=2$，得图1-28所示程序。最后，我们看到，图1-28中第二个$L:=1$是不必要的，因为已有初始化$L:=1$。删去第二个$L:=1$后得最后的结构程序如图1-29所示。我们称这